ENG ME702: Computational Fluid Mechanics (Fall 2016)

Final Project

Shock Tube with *rhoCentralFoam*, *sonicFoam*, Lax-Friedrichs and MacCormacks

Ankush Gupta (U01997897)

Submitted on: December 20, 2016

Submitted to: Dr. Sheryl Grace

Department of Mechanical Engineering

College of Engineering

Boston University

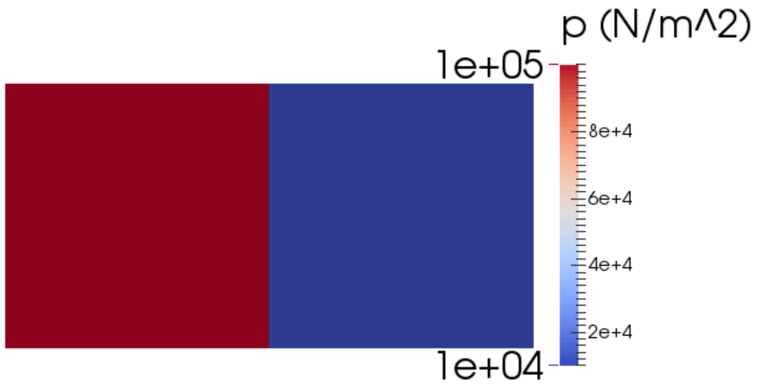
Content

1. Description of the problem

In this project, an analysis on the shock tube was performed. During the analysis, a comparison was created between the effect of viscosity (μ) in a two-dimensional system. Two different solvers, namely, rhoCentralFoam and sonicFoam were analyzed. Also, a comparison between the rhoCentralFoam solver and Lax-Friedrichs and MacCormack algorithms was performed.

2. Hypotheses

- Compressible flow
- Viscous flow
- Bi-dimensional flow $\left(\frac{\partial}{\partial z} = 0\right)$
- Laminar



3. Shock Tube with rhoCentralFoam, sonicFoam, Lax-Friedrichs and MacCormacks

3.1. Pre-processing

To start openFoam on the supercomputer, we follow the following command.

```
module load openfoam
```

For the initial setup of the files, I transferred the basic setup from of the Shock Tube problem from the sonicFoam and rhoCentralFoam solvers. For that, I used the following command.

```
cp -r $FOAM_TUTORIALS/compressible/rhoCentralFoam/shockTube/ $FOAM_RUN/rhoCentralFoam/shockTube
```

&

```
cp -r $FOAM_TUTORIALS/compressible/sonicFoam/laminar/shockTube/ $FOAM_RUN/rhoCentralFoam/shockTube
```

3.1.1. Mesh generation

After the setup, the mesh is defined and refined to the specific comparison. For that, we choose a domain of 10*5*2 meters. The mesh size was chosen to be 300*150*1 where the refinement was conducted near the initial dam of the domain.

To edit the mesh file, following code was used.

```
nedit shockTube/constant/polyMesh/blockMeshDict
```

In the editor, the code was edited. The mesh filed used is as shown below.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

convertToMeters 1;

vertices
(
    (-5 -2.5 -1)
    (5 -2.5 -1)
    (5 2.5 -1)
    (-5 2.5 -1)
    (-5 -2.5 1)
    (5 -2.5 1)
    (5 2.5 1)
    (-5 2.5 1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (300 150 1)
    simpleGrading
    (

      (
           (30 22.5 0.5)
           (30 55 1)
           (35 22.5 2)
      )
      1
      1
     )
);

edges
(
);

boundary
(
    sides
    {
        type patch;
        faces
        (
            (1 2 6 5)
            (0 4 7 3)
        );
    }
    topandbottom
    {
     type wall;
     faces
     (
         (0 1 5 4)
         (3 7 6 2)
      );

    }

    empty
    {
        type empty;
        faces
        (
            (5 6 7 4)
            (0 3 2 1)
        );
    }
);

mergePatchPairs
(
);

//
**********************************************************************
*** //
```

3.1.2. Boundary and initial conditions

We setup the boundary conditions in the file. For that we update the *\p*, *\T* and *\U* directories in the *\0* folder.

The pressure in the *\0\p* directory file is as follows.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    sides
    {
        type            zeroGradient;
    }
    topandbottom
    {
     type       zeroGradient;
    }

    empty
    {
        type            empty;
    }
}
```

```
//
**********************************************************************
*** //
```

The temperature in *\0\T* directory is as follows.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 1;

boundaryField
{
    sides
    {
        type            zeroGradient;
    }
    topandbottom
    {
        type            zeroGradient;
    }

    empty
    {
        type            empty;
    }
}
```

```
//
**********************************************************************
*** //
```

The velocity in the *0* *U* directory as follows.

```
/*---------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*----------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    sides
    {
        type            zeroGradient;
    }
    topandbottom
    {
        type            zeroGradient;
    }
    empty
    {
        type            empty;
    }
}
```

```
//
***********************************************************************
*** //
```

The boundary conditions files reported above are for the *rhoCentralFoam* solver. In the *sonicFoam* case, an additional file *\0\magU* needs to be updated. The file looks like below.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      magU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    sides
    {
        type            calculated;
        value           uniform 0;
    }
    topandbottom
    {
        type            calculated;
        value           uniform 0;
    }
    empty
    {
        type            empty;
    }
}

//
**********************************************************************
*** //
```

After the setup of the boundary condition files, we set the initial conditions in the *\system\setFieldsDict* directory which is shown as below.

```
/*--------------------------------*- C++ -*---------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*--------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

defaultFieldValues ( volVectorFieldValue U ( 0 0 0 )
volScalarFieldValue T 348.432 volScalarFieldValue p 100000 );

regions         ( boxToCell { box ( 0 -2.5 -1 ) ( 5 2.5 1 ) ;
fieldValues ( volScalarFieldValue T 278.746 volScalarFieldValue p
10000 ) ; } );


//
**********************************************************************
*** //
```

3.1.3. Physical properties

The physical properties of the case are stored in the dictionaries whose names are given the suffix ..Properties. The *thermophysicalProperties* files looks as shown below. In this file, the value of the μ is chosen by taking the average of the viscosity for the domain over the two different initial conditions. The calculation shows that the value of the viscosity is 1.932e-5.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        nMoles          1;
        molWeight       28.96;
    }
    thermodynamics
    {
        Cp              1004.5;
        Hf              2.544e+06;
    }
    transport
    {
        mu              1.932e-05;
        Pr              1;
    }
}


//
***********************************************************************
*** //
```

The \constant\turbulenceProperties\\ as follows.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

simulationType  laminar;


//
***********************************************************************
*** //
```

3.1.4. Control

Input data related to the control of the time and reading and writing of the solution data are read in from the *\system\controlDict* dictionary. The file looks like the following. This file helps in choosing the solver as well.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     rhoCentralFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.007;

deltaT          1e-06;

writeControl    adjustableRunTime;

writeInterval   0.001;

cycleWrite      0;

writeFormat     ascii;
```

```
writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           0.2;

maxDeltaT       1;


//
**********************************************************************
*** //
```

3.1.5. Discretization and linear-solver setting

There must be two more files in the *\system* directory. The first is called *fvSchemes* and specifies the choice of finite volume discretization schemes (for each one of the terms of the differential equations governing the problem). The second is called *fvSolution* and contains the specifications of the linear equations solvers and tolerances and other algorithm controls. The dictionary *\system\fvSchemes* look as below.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

fluxScheme      Kurganov;

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(tauMC)      Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    reconstruct(rho) vanLeer;
    reconstruct(U)  vanLeerV;
    reconstruct(T)  vanLeer;
}

snGradSchemes
{
    default         corrected;
}


//
**********************************************************************
*** //
```

And the dictionary *system*\fvSolution is as follows.

```
/*--------------------------------*- C++ -*----------------------------
-------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|
|  \\    /   O peration     | Version:  2.4.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------
-------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

solvers
{
    "(rho|rhoU|rhoE)"
    {
        solver          diagonal;
    }

    "(U|e)"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-09;
        relTol          0.01;
    }

    h
    {
        $U;
        tolerance       1e-10;
        relTol          0;
    }
}


//
*********************************************************************
*** //
```

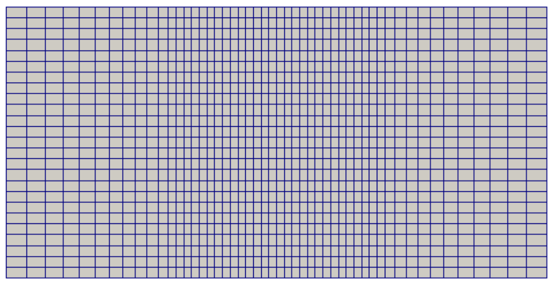
Up to now, all the required fields and dictionaries to run the case have been set. As a final summary and if all the steps have been followed properly, the used should have the following scheme in the shockTube directory for the *rhoCentralFoam* and *sonicFoam* case.

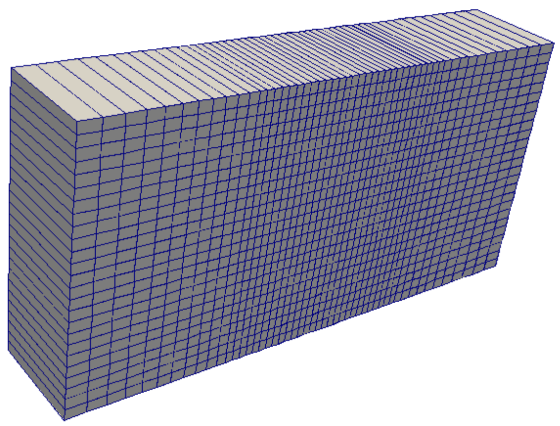
$$shockTube\_rhoCentralFoam\left\{\begin{array}{l} 0\left\{\begin{array}{l} p\\ T\\ U \end{array}\right.\\ constant\left\{\begin{array}{c} polyMesh\{blockMeshDict\\ thermophysicalProperties\\ turbulenceProperties \end{array}\right.\\ system\left\{\begin{array}{l} controlDict\\ fvSchemes\\ fvSolution\\ sampleDict\\ setFieldsDict \end{array}\right. \end{array}\right.$$

$$shockTube\_sonicFoam\left\{\begin{array}{l} 0\left\{\begin{array}{l} p\\ T\\ U\\ magU \end{array}\right.\\ constant\left\{\begin{array}{c} polyMesh\{blockMeshDict\\ thermophysicalProperties\\ turbulenceProperties \end{array}\right.\\ system\left\{\begin{array}{l} controlDict\\ fvSchemes\\ fvSolution\\ sampleDict\\ setFieldsDict \end{array}\right. \end{array}\right.$$

3.1.6. Creating the mesh

The mesh is generated after executing the command *blockMesh*. Running the *paraFoam* command lets one view the mesh which has been shown below.





3.2.Running the application

To run the program, first we execute the command *setFields*. This command refers the *\system\setFieldsDict* and update the dictionaries in the directory *\0* in accordance with the mesh size to assign the initial value.

Then, we run the solver. That is executed by calling the name of the solver i.e. *rhoCentralFoam* or *sonicFoam* depending upon the case.

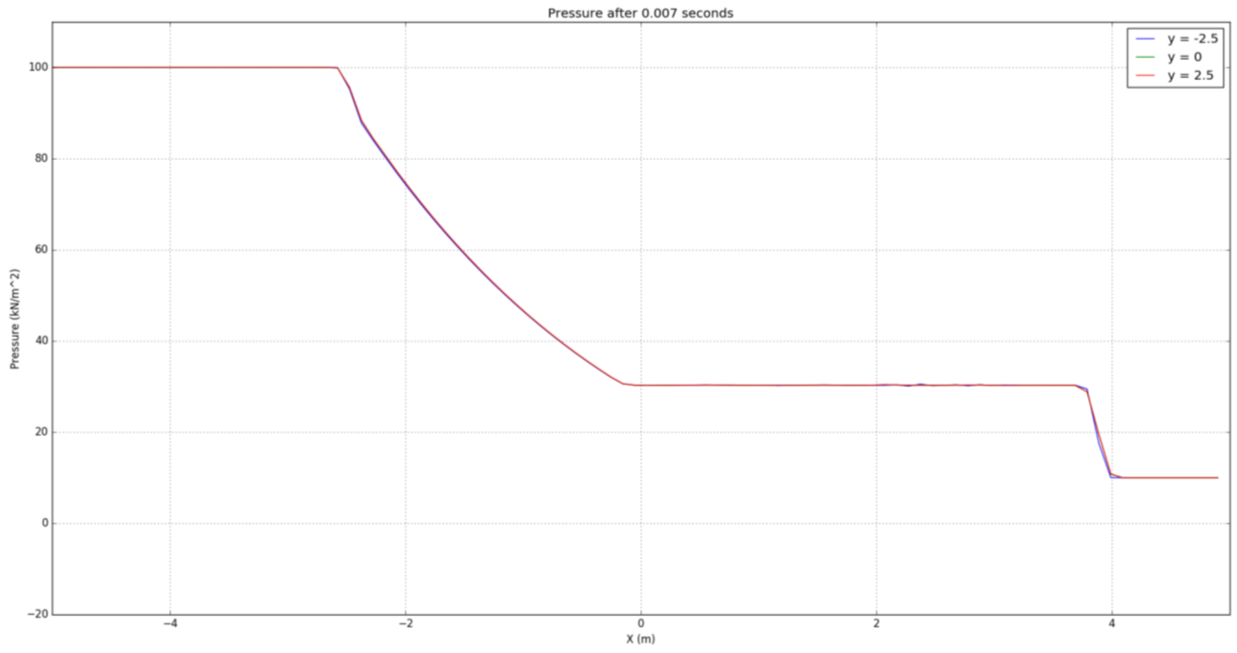
After running the solver, we can view the results in ParaView by executing the command *paraFoam*.

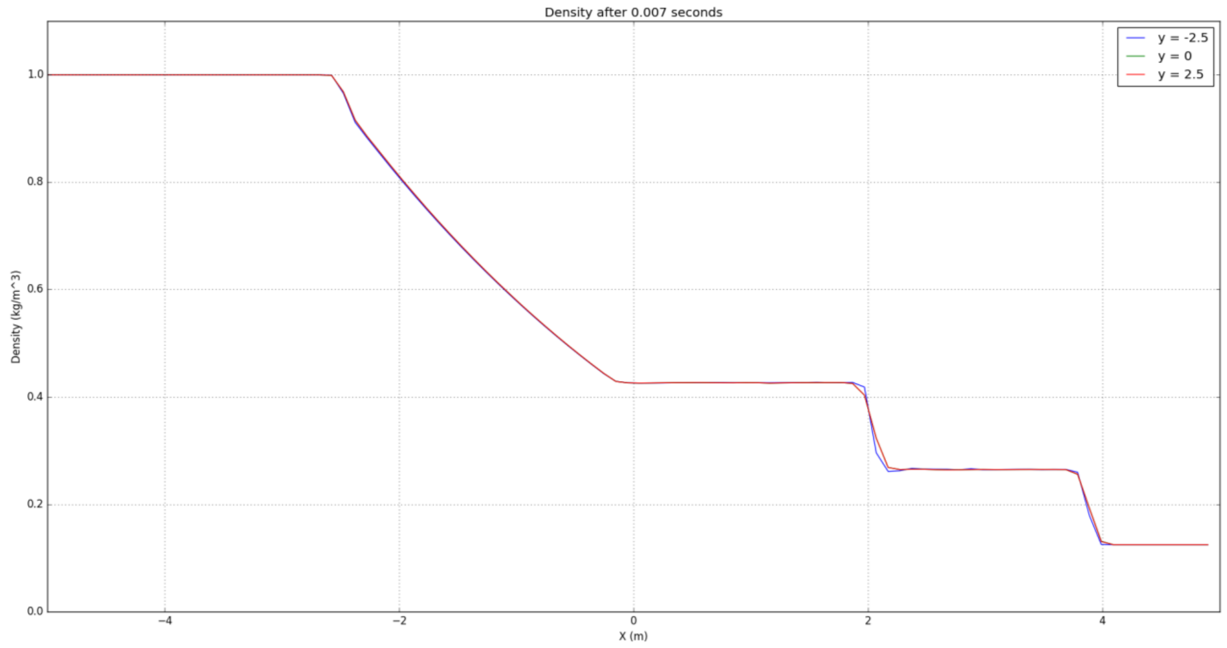
To obtain the values for postprocessing, we run the command *sample -case ..\shockTube*. This company refers to the dictionary *\system\sampleDict* and deposits the results in the *\postProcessing* directory.

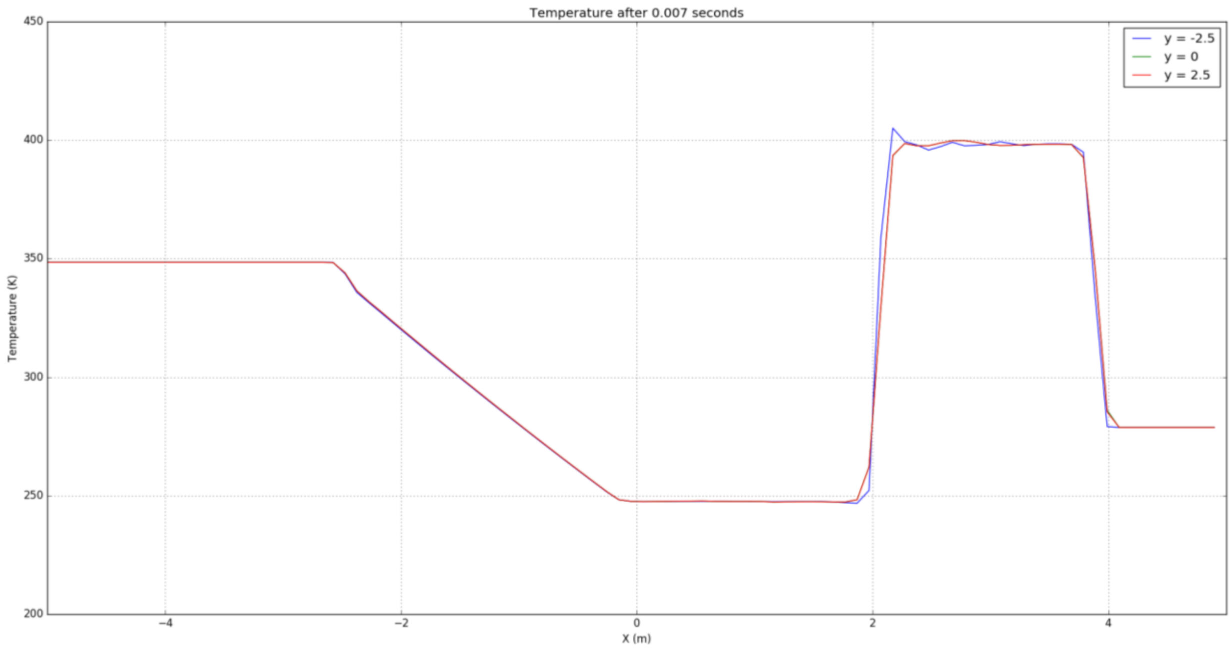
4. Results & Discussion

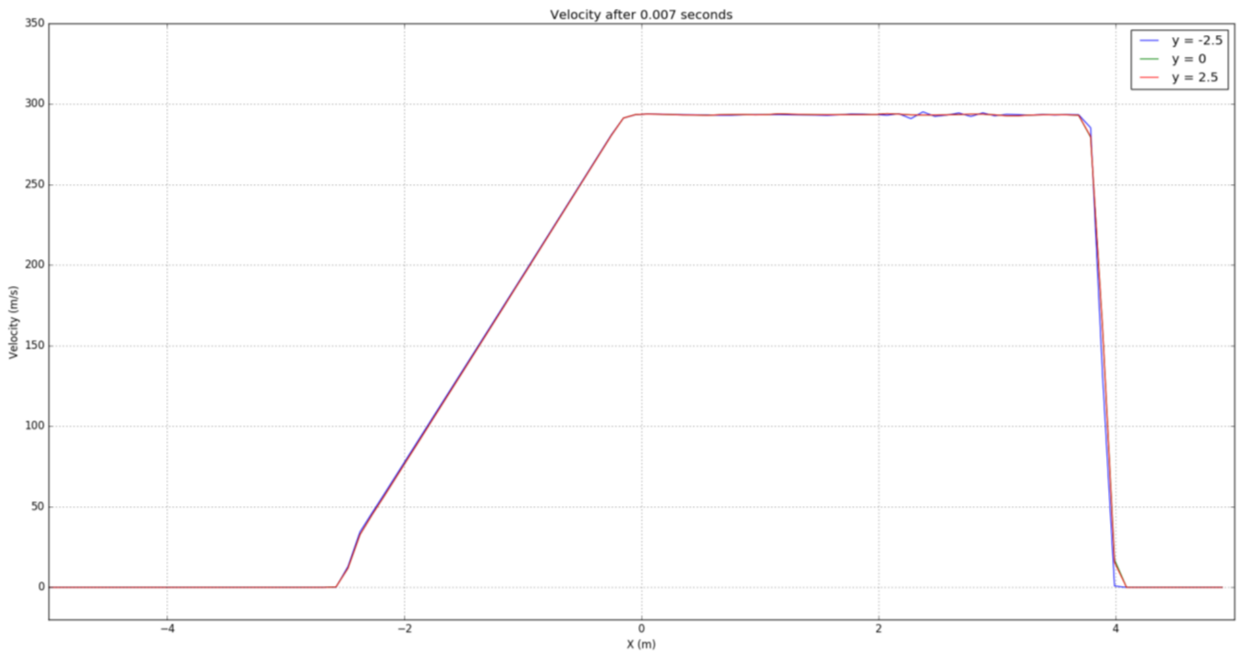
4.1.*rhoCentralFoam*

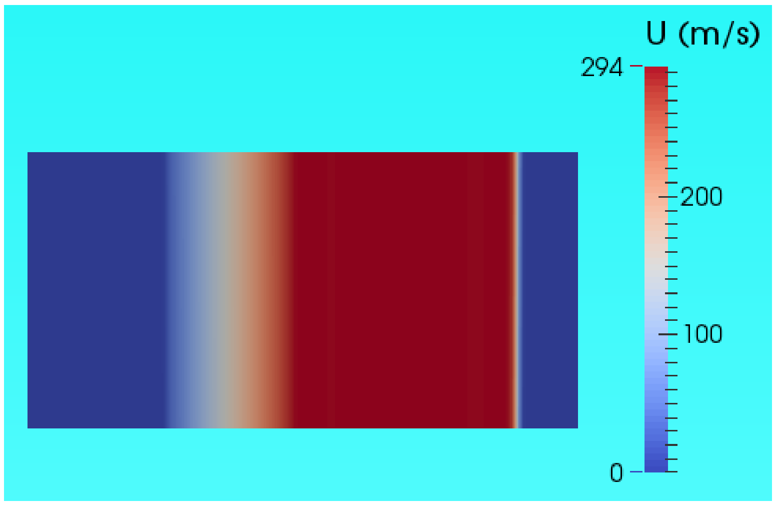
Comparative results while solving the shock tube problem using the *rhoCentralFoam* solver in OpenFOAM are shown below with the physical properties measured at three different points in the y-axis.

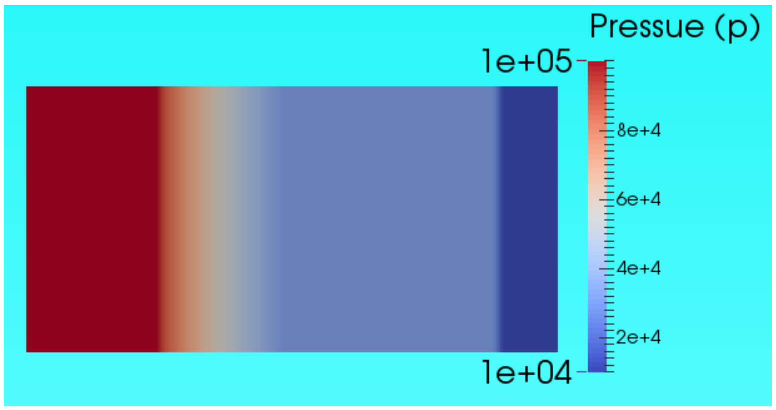



Density after 0.007 seconds
Density (kg/m^3)
X (m)
y = -2.5
y = 0
y = 2.5
1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4


Temperature after 0.007 seconds
Temperature (K)
X (m)
y = -2.5
y = 0
y = 2.5
450
400
350
300
250
200
−4
−2
0
2
4





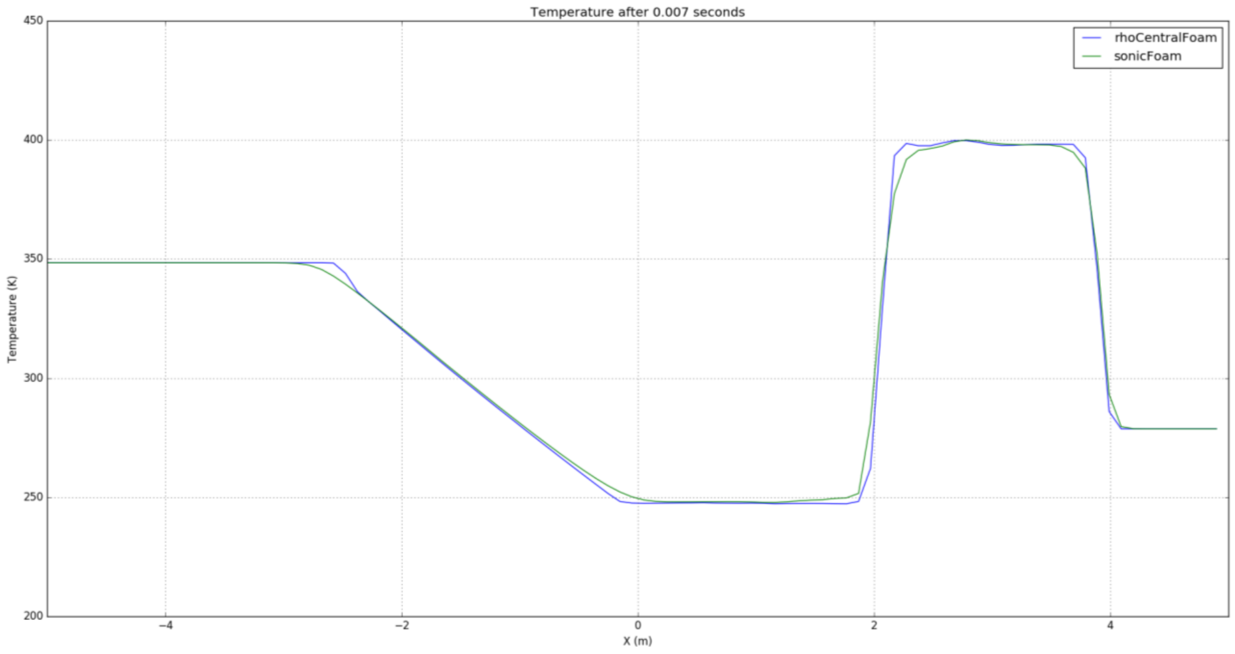


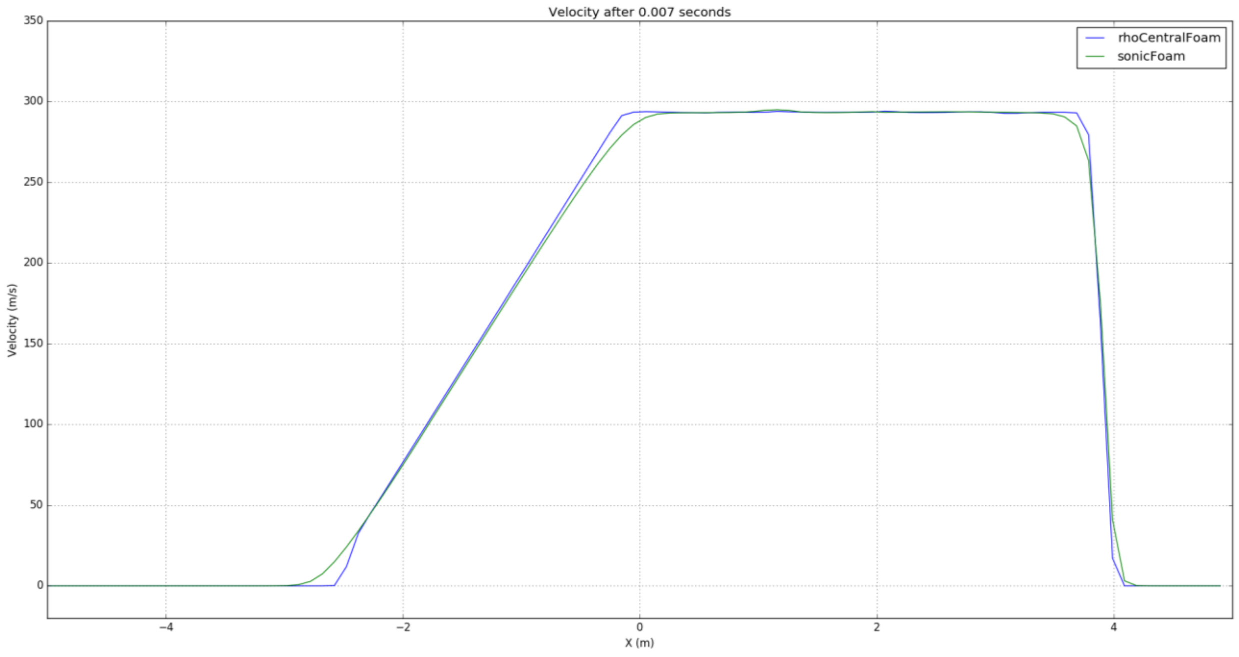
<u>Discussion:</u>

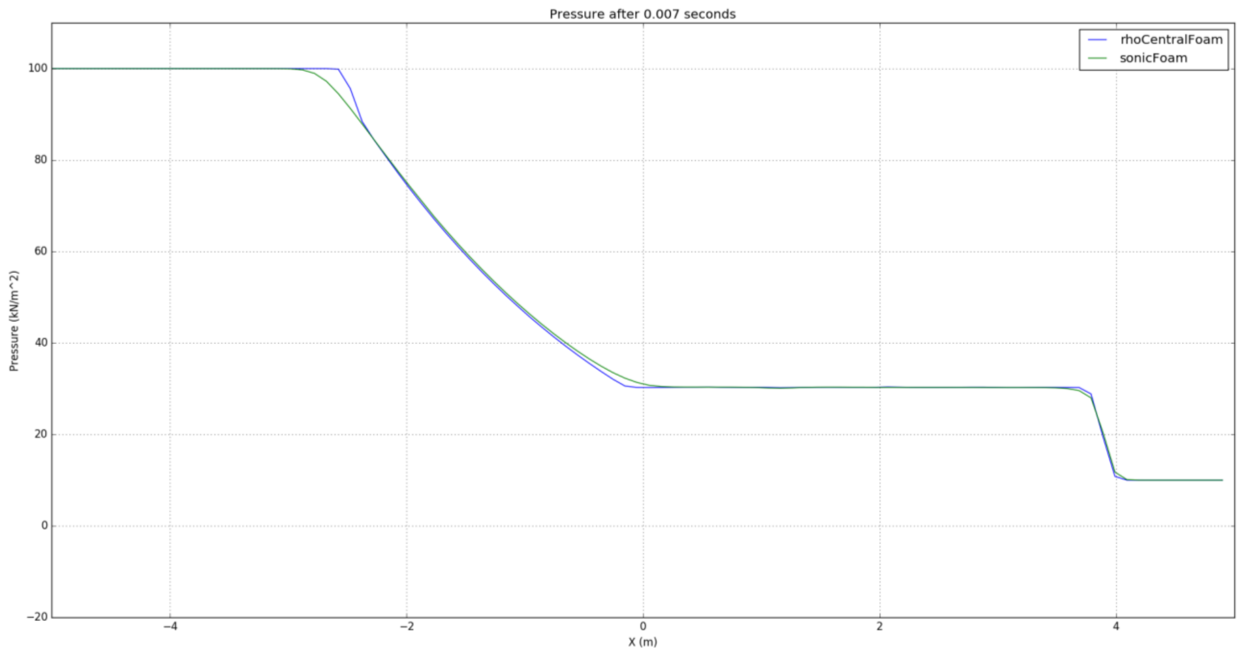
These curves were plotted to see the changes observed in the system because of adding viscosity to the physical properties of the fluid inside. From the curves plotted above, we can observe relatively higher oscillations when the domain under question is closer to the bottom plate (i.e. y = -2.5 m) as compared to the rest of the domain. We can also observe that there is no difference in the physical properties from when y = 0 m and y = +2.5m.

4.2.*sonicFoam* vs *rhoCentralFoam*

Here, a comparison has been made for results obtained for the same problem by two different solvers in OpenFOAM. The solvers under question are *sonicFoam* and *rhoCentralFoam*. The plots plotted below are for the domain with a horizontal line with a y = 0 m. These plots were plotted at 7 milliseconds after rupturing the hypothetical wall.







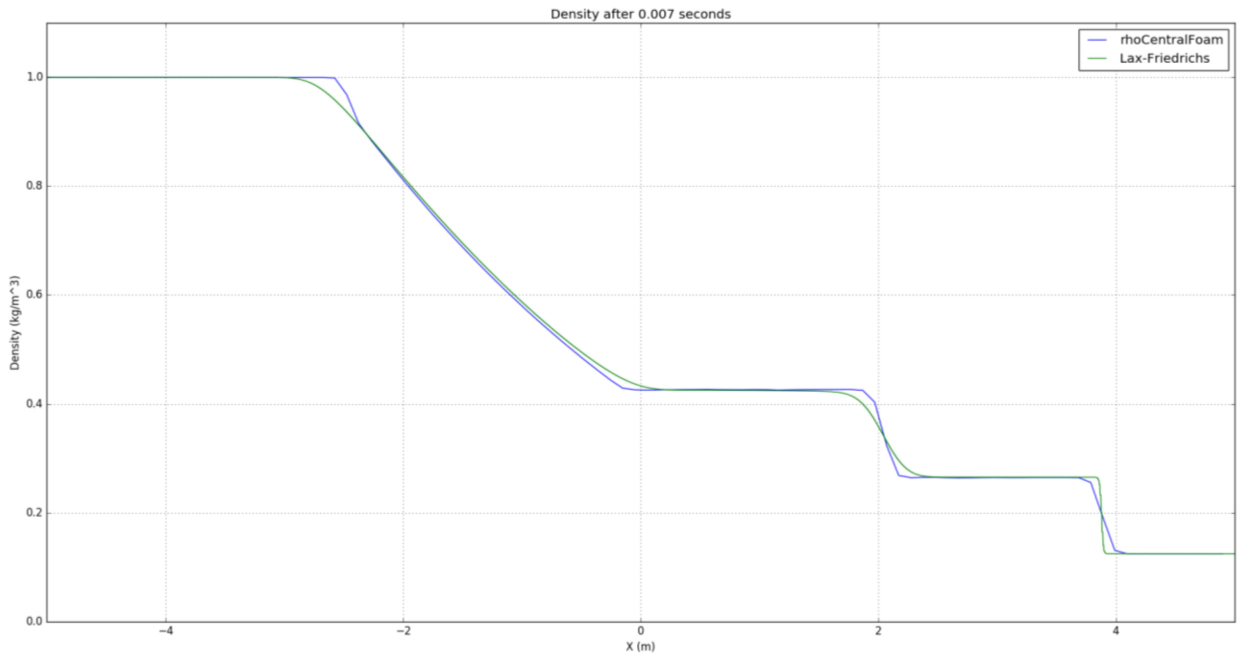
<u>Discussion:</u>

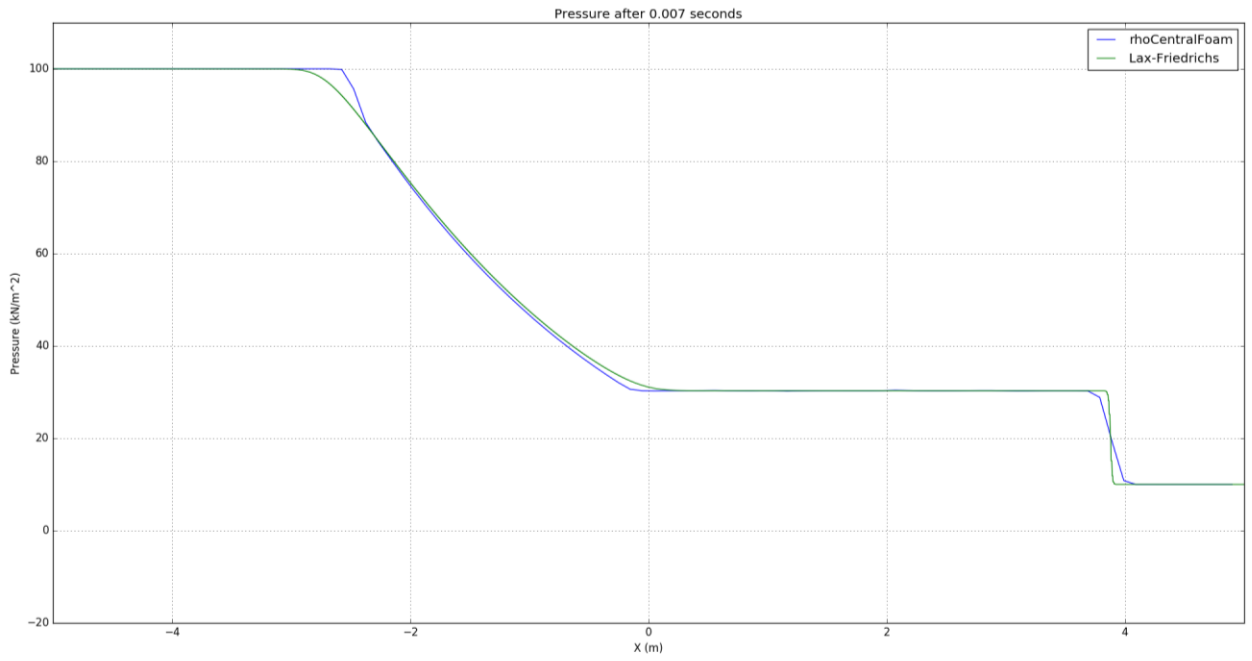
From the comparison of the pressure, temperature and velocity plots obtained by plotting the results for both the solvers, we observe that for *rhoCentralFoam* solver, the curves are sharper. The magnitude of the slopes at the critical points are higher which are closer to the analytical solution, hence results obtained for shock tube problem from *rhoCentralFoam* are better than the results obtained by using *sonicFoam* solver. Adding to that, we also saw that there was a considerable difference in the amount of time taken by sonicFoam to reach the solution (not quite as accurate to the ones obtained by using *rhoCentralFoam* solver). The time taken by *sonicFoam* solver was 139 seconds whereas for *rhoCentralFoam* solver was 60 seconds.

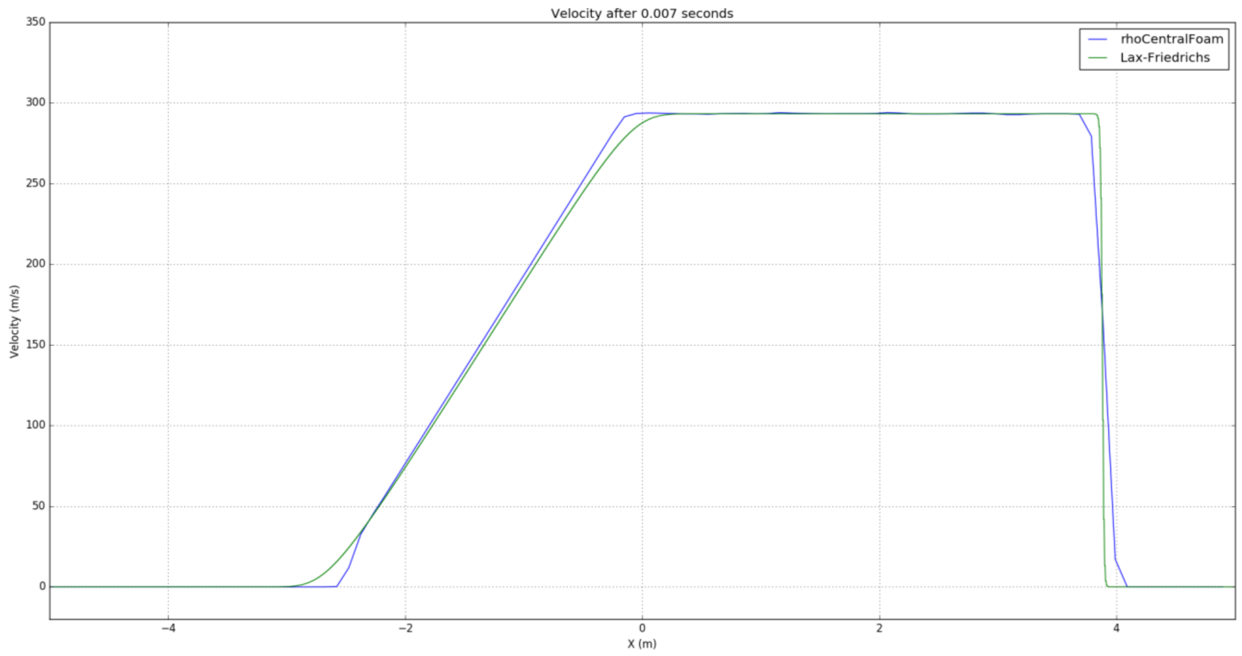
This clearly demarcates that *rhoCentralFoam* is a better solver than *sonicFoam* for problems like the shock tube.

4.3.*rhoCentralFoam* vs Lax-Friedrichs

Here, the results obtained from *rhoCentralFoam* solver have been compared with the results obtained by a self-written code (Appendix A.1.) in python which follows the Lax-Friedrichs(LF) algorithm.







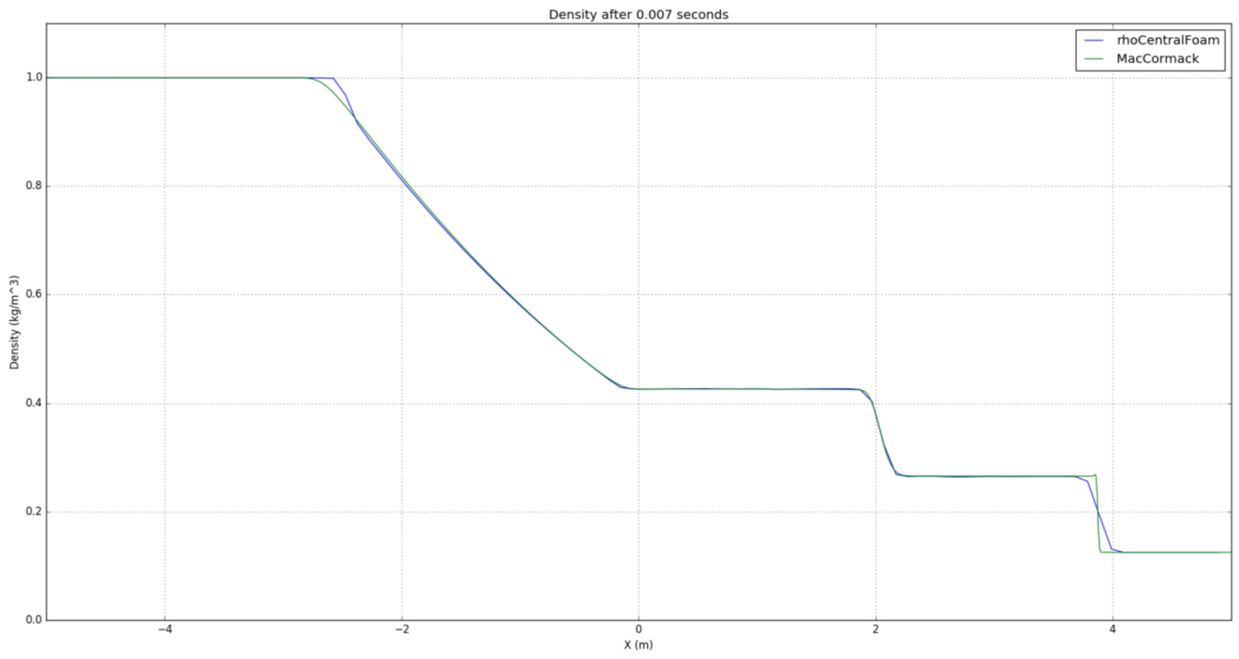
<u>Discussion:</u>

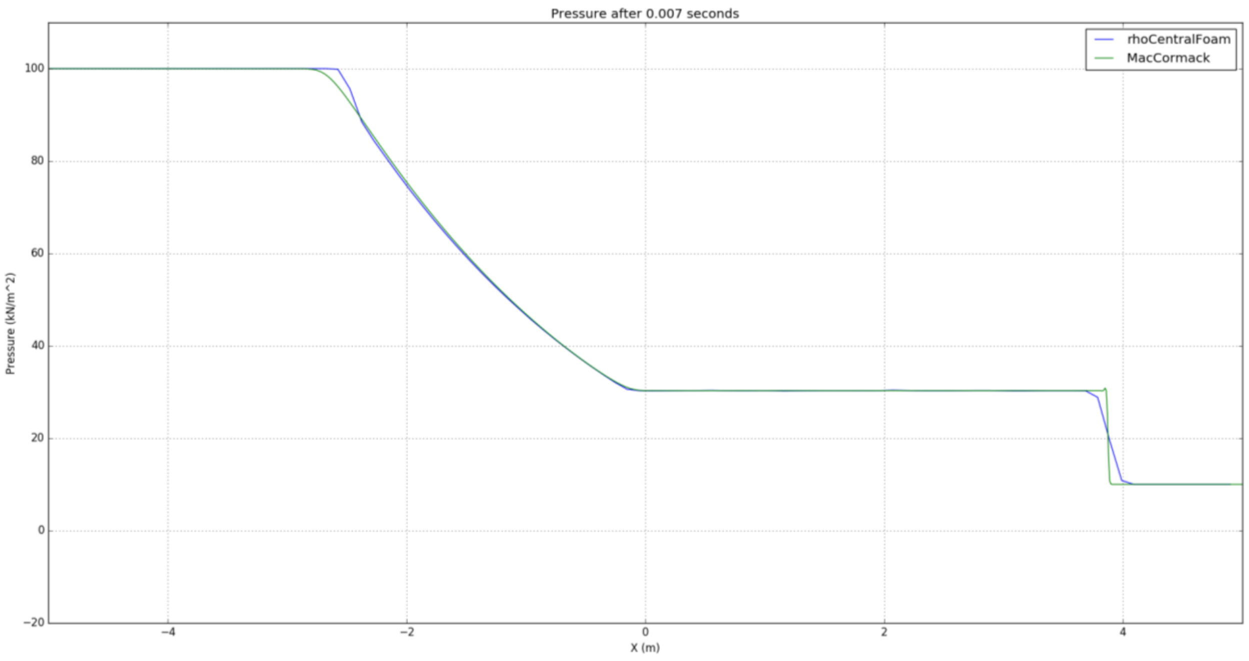
From the curves plotted above, we see that LF code performs better for some instances, for example the vertical shock drops, as compared to the curves obtained by using *rhoCentralFoam* solver. Something we can observe that the OpenFOAM solver performs between in portions of gradual change. As we will see in the next case, they match quite well.

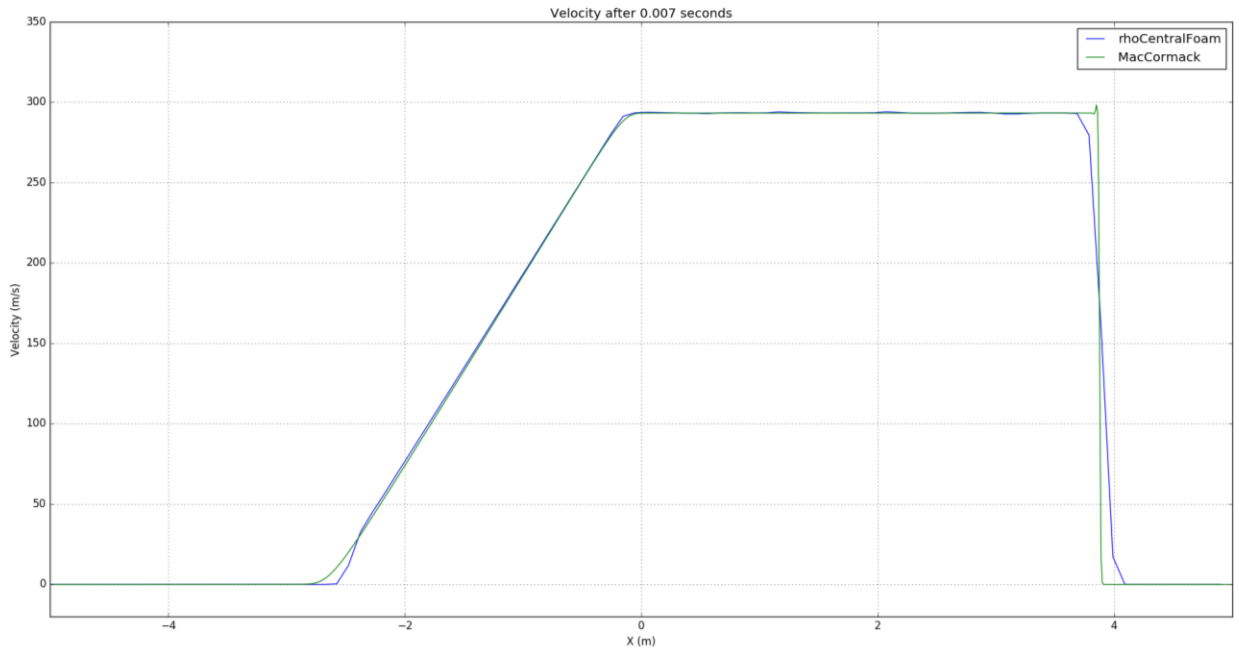
Disclaimer: By increasing the mesh size of the domain or by increasing the sampling point, we may obtain better results while using *rhoCentralFoam* solver.

4.4. *rhoCentralFoam* vs MacCormack

Here, the results obtained from *rhoCentralFoam* solver have been compared with the results obtained by a self-written code (Appendix A.2.) in python which follows the MacCormack (MC) algorithm.







Discussion:

From the curves plotted above, we can see that the *rhoCentralFoam* solver follows much closer to the MC solution. And from theory, we understand that MC is a much better way to approach a complicated problem like the shockTube as compared to using the LF algorithm. We also notice that MC, much like the LF has a much sharper slope at the extremely high slope areas as compared to the *rhoCentralFoam* solver, which might be resolved by refining the mesh a bit more, or by increasing the number of sampling points.

Also, something worth noticing is that the codes written in python were much faster to simulate the results, BUT, we should also keep in mind that while solving the program using python, the domain was one dimensional, whereas, it was a two-dimensional domain for OpenFOAM. Also, OpenFOAM is more robust, and functional than performing fluid dynamic simulations in python. The process for initial setup for complex problems and geometries will be extensive in python as compared to OpenFOAM.

Appendix

A.1. Python code for Lax-Fredrichs and plotting results in comparison with OpenFOAM results.

```
def f_f(uu,nn):
    ff = np.zeros((nn,3))
    ff[:,0] = uu[:,1]
    ff[:,1] = (uu[:,1]*uu[:,1]/uu[:,0]) + (gamma - 1)*(uu[:,2] - 
uu[:,1]*uu[:,1]/(2*uu[:,0]))
    ff[:,2] = (uu[:,2] + (gamma - 1)*(uu[:,2] - 
uu[:,1]*uu[:,1]/(2*uu[:,0])))*(uu[:,1]/uu[:,0])
    return ff

import numpy as np
import openpyxl
import matplotlib.pyplot as pyplot


#Constants
xmin = -5
xmax = 5
ne = 2500
nn = ne +1


xsteps = np.linspace(xmin,xmax,nn)
dx = (xmax-xmin)/ne

sigma = 0.4 #CFL
gamma = 1.4  #Taken from the book
c = 374.17 #m/s : Initial wave speed


dt = sigma*dx/c
T = 0.007

nt = int(T/dt) + 1




#INITIAL CONDITIONS
rho_L = 1
u_L = 0
p_L = 100000

rho_R = 0.125
u_R = 0
p_R = 10000
```

```
#Intial Flux and basic vectors

uu = np.zeros((nn,3)) # uu is the basic vector which is euqal to [rho; 
rho*u ; rho*e_t]
ff = np.zeros((nn,3))
#Initializaition
for i in range(nn):

    if i < int(nn/2):
        uu[i,0] = rho_L
        uu[i,1] = rho_L*u_L
        uu[i,2] = rho_L * ( p_L/((gamma-1)*rho_L) + u_L*u_L*0.5)
        ff[i,0] = uu[i,1]
        ff[i,1] = ff[i,0]*(uu[i,1]/uu[i,0]) + p_L
        ff[i,2] = ff[i,0]*(uu[i,2]/uu[i,0] + 
(uu[i,1]/uu[i,0])*(uu[i,1]/uu[i,0])*0.5)

    else:
        uu[i,0] = rho_R
        uu[i,1] = rho_R*u_R
        uu[i,2] = rho_R * ( p_R/((gamma-1)*rho_R) + u_R*u_R*0.5)
        ff[i,0] = uu[i,1]
        ff[i,1] = ff[i,0]*(uu[i,1]/uu[i,0]) + p_R
        ff[i,2] = ff[i,0]*(uu[i,2]/uu[i,0] + 
(uu[i,1]/uu[i,0])*(uu[i,1]/uu[i,0])*0.5)

uu_t = np.zeros((nn,3))


for t in range(nt):
    uu_t[1:-1,:] = 0.5*(uu[2:,:]+uu[:-2,:]) - dt*0.5*(ff[2:,:]-ff[:-
2,:])/dx
    uu_t[0,:] = uu_t[1,:]
    uu_t[-1,:] = uu_t[-2,:]
    ff = f_f(uu_t,nn)
    uu = uu_t.copy()


wb = openpyxl.load_workbook('0.004.xlsx')


p = np.zeros((100,4))
u = np.zeros((100,4))
T = np.zeros((100,4))
rho = np.zeros((100,4))

for i in range (1,100):
    sheet = wb.get_sheet_by_name('0.007_00')
    p[i,0] = sheet['A' + str(i+1)].value
    u[i,0] = sheet['A' + str(i+1)].value
    T[i,0] = sheet['A' + str(i+1)].value
    rho[i,0] = sheet['A' + str(i+1)].value
    sheet = wb.get_sheet_by_name('0.007_00')

    p[i,2] = sheet['B' + str(i+1)].value
    u[i,2] = sheet['G' + str(i+1)].value
    T[i,2] = sheet['C' + str(i+1)].value
    rho[i,2] = sheet['D' + str(i+1)].value




pyplot.figure(1)
pyplot.plot(u[:,0]-5, u[:,2],label ="rhoCentralFoam")
pyplot.plot(xsteps,uu[:,1]/uu[:,0],label = 'Lax-Friedrichs')
pyplot.axis([-5,5,-20,350])
pyplot.title('Velocity after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Velocity (m/s)')
pyplot.grid(True)
pyplot.legend()

pyplot.figure(2)
pyplot.plot(p[:,0]-5, p[:,2]/1000,label ="rhoCentralFoam")
pyplot.plot(xsteps,(ff[:,1] - uu[:,1]*uu[:,1]/uu[:,0])/1000,label =
'Lax-Friedrichs')
pyplot.axis([-5,5,-20,110])
pyplot.title('Pressure after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Pressure (kN/m^2)')
pyplot.grid(True)
pyplot.legend()
pyplot.show()


pyplot.figure(3)
pyplot.plot(rho[:,0]-5, rho[:,2],label ="rhoCentralFoam")
pyplot.plot(xsteps,uu[:,0],label = 'Lax-Friedrichs')
pyplot.axis([-5,5,0,1.1])
pyplot.title('Density after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Density (kg/m^3)')
pyplot.grid(True)
pyplot.legend()
pyplot.show()
```

A.2. Python code for MacCormacks and plotting results in comparison with OpenFOAM results.

```
def f_f(uu,nn):
    ff = np.zeros((nn,3))
    ff[:,0] = uu[:,1]
    ff[:,1] = (uu[:,1]*uu[:,1]/uu[:,0]) + (gamma - 1)*(uu[:,2] -
uu[:,1]*uu[:,1]/(2*uu[:,0]))
    ff[:,2] = (uu[:,2] + (gamma - 1)*(uu[:,2] -
uu[:,1]*uu[:,1]/(2*uu[:,0])))*(uu[:,1]/uu[:,0])
    return ff

import numpy as np
import openpyxl
import matplotlib.pyplot as pyplot
#Constants
xmin = -5
xmax = 5
ne = 1000
nn = ne +1


xsteps = np.linspace(xmin,xmax,nn)
dx = (xmax-xmin)/ne

sigma = 0.4 #CFL
gamma = 1.4  #Taken from the book
c = 374.17 #m/s : Initial wave speed
epi = 0.04

dt = sigma*dx/c
T = 0.007

nt = int(T/dt) + 1


#INITIAL CONDITIONS
rho_L = 1
u_L = 0
p_L = 100000

rho_R = 0.125
u_R = 0
p_R = 10000

#Intial Flux and basic vectors

uu = np.zeros((nn,3)) # uu is the basic vector which is euqal to [rho;
rho*u ; rho*e_t]
ff = np.zeros((nn,3))
#Initializaition
```

```
for i in range(nn):

    if i < int(nn/2):
        uu[i,0] = rho_L
        uu[i,1] = rho_L*u_L
        uu[i,2] = rho_L * ( p_L/((gamma-1)*rho_L) + u_L*u_L*0.5)
        ff[i,0] = uu[i,1]
        ff[i,1] = ff[i,0]*(uu[i,1]/uu[i,0]) + p_L
        ff[i,2] = ff[i,0]*(uu[i,2]/uu[i,0] +
(uu[i,1]/uu[i,0])*(uu[i,1]/uu[i,0])*0.5)

    else:
        uu[i,0] = rho_R
        uu[i,1] = rho_R*u_R
        uu[i,2] = rho_R * ( p_R/((gamma-1)*rho_R) + u_R*u_R*0.5)
        ff[i,0] = uu[i,1]
        ff[i,1] = ff[i,0]*(uu[i,1]/uu[i,0]) + p_R
        ff[i,2] = ff[i,0]*(uu[i,2]/uu[i,0] +
(uu[i,1]/uu[i,0])*(uu[i,1]/uu[i,0])*0.5)

uu_t = np.zeros((nn,3))
uu_m = np.zeros((nn,3))


for t in range(nt):
    uu_m[1:-1,:] = uu[1:-1,:] - dt*(ff[2:,:]-ff[1:-1,:])/dx #+
epi*(uu[2:,:] -2*uu[1:-1,:] + uu[:-2,:])
    uu_m[0,:] = uu_m[1,:]
    uu_m[-1,:] = uu_m[-2,:]
    ff = f_f(uu_m,nn)
    uu_t[1:-1,:] = 0.5*(uu[1:-1,:] + uu_m[1:-1,:]) -dt*0.5*(ff[1:-
1,:]-ff[:-2,:])/dx + epi*(uu[2:,:] -2*uu[1:-1,:] + uu[:-2,:])
    uu_t[0,:] = uu_t[1,:]
    uu_t[-1,:] = uu_t[-2,:]
    ff = f_f(uu_t,nn)
    uu = uu_t.copy()

wb = openpyxl.load_workbook('0.004.xlsx')


p = np.zeros((100,4))
u = np.zeros((100,4))
T = np.zeros((100,4))
rho = np.zeros((100,4))

for i in range (1,100):
    sheet = wb.get_sheet_by_name('0.007_00')
    p[i,0] = sheet['A' + str(i+1)].value
    u[i,0] = sheet['A' + str(i+1)].value
    T[i,0] = sheet['A' + str(i+1)].value
    rho[i,0] = sheet['A' + str(i+1)].value
    sheet = wb.get_sheet_by_name('0.007_00')

    p[i,2] = sheet['B' + str(i+1)].value
    u[i,2] = sheet['G' + str(i+1)].value
    T[i,2] = sheet['C' + str(i+1)].value
    rho[i,2] = sheet['D' + str(i+1)].value




pyplot.figure(1)
pyplot.plot(u[:,0]-5, u[:,2],label ="rhoCentralFoam")
pyplot.plot(xsteps,uu[:,1]/uu[:,0],label = 'MacCormack')
pyplot.axis([-5,5,-20,350])
pyplot.title('Velocity after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Velocity (m/s)')
pyplot.grid(True)
pyplot.legend()

pyplot.figure(2)
pyplot.plot(p[:,0]-5, p[:,2]/1000,label ="rhoCentralFoam")
pyplot.plot(xsteps,(ff[:,1] - uu[:,1]*uu[:,1]/uu[:,0])/1000,label =
'MacCormack')
pyplot.axis([-5,5,-20,110])
pyplot.title('Pressure after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Pressure (kN/m^2)')
pyplot.grid(True)
pyplot.legend()
pyplot.show()


pyplot.figure(3)
pyplot.plot(rho[:,0]-5, rho[:,2],label ="rhoCentralFoam")
pyplot.plot(xsteps,uu[:,0],label = 'MacCormack')
pyplot.axis([-5,5,0,1.1])
pyplot.title('Density after 0.007 seconds')
pyplot.xlabel('X (m)')
pyplot.ylabel('Density (kg/m^3)')
pyplot.grid(True)
pyplot.legend()
pyplot.show()
```